# Programming Assignment 2: Airport Concurrency

## Description

In this assignment you will gain hands-on experience with parallel programming and the difficulty of building correct parallel programs. You are tasked with helping an airport manage its single runway. The airport serves both commercial flights, cargo flights, and occasional emergency aircraft, and must carefully coordinate runway access to ensure safety and efficiency. The air traffic controller manages a high volume of aircraft operations with various constraints including runway direction changes, fuel limitations, and emergency priorities. Several restrictions must be imposed to maintain safety and controller wellbeing.

## Functional Requirements

**Requirement 1:** The runway can only accommodate 2 aircraft at a time (one preparing for takeoff, one in final approach). No more than 2 aircraft are allowed to simultaneously use the runway. When the runway is full and new aircraft arrive they must wait in a holding pattern.

**Requirement 2:** Air traffic control gets confused when managing commercial and cargo flights simultaneously. The controller decides that while commercial flights are using the runway, no cargo flights are allowed to use it, and vice versa.

**Requirement 3:** The air traffic controller gets fatigued after managing too many operations. The controller decides that after managing 8 aircraft operations, a break is needed before managing more aircraft. So after the 8th aircraft (counting since the last break) uses the runway, no more aircraft are permitted to use the runway until after the controller's next break. Aircraft that arrive while the controller is on break must enter a holding pattern.

**Requirement 4:** In order to be fair to both flight types, after 4 consecutive aircraft of a single type, the controller will give priority to an aircraft of the other type.

**Requirement 5:** Your program should ensure progress, i.e. if the runway is empty and the controller is not currently on break, an arriving aircraft should not be forced to wait (unless a direction switch is required and the aircraft needs the opposite direction). Similarly, if an arriving aircraft is compatible with the aircraft currently using the runway (same direction, same type for commercial/cargo, and runway not at capacity), it should not be forced to wait, unless the controller is due for a break or higher-priority aircraft are waiting.

**Requirement 6:** Emergency aircraft can arrive randomly and must be given high priority. When an emergency aircraft arrives, it must be granted runway access within 30 seconds. Emergency aircraft can interrupt the normal rotation between commercial and cargo flights, but still cannot violate the controller's break schedule or the runway direction rules. If multiple aircraft are

waiting, emergency aircraft should be prioritized, but the system must still ensure that regular flights are not starved indefinitely.

**Requirement 7:** The runway can be used in two directions: North and South. Commercial aircraft prefer the North direction, while cargo aircraft prefer the South direction. Emergency aircraft can use either direction. However, the runway can only operate in one direction at a time. To change direction, the runway must be completely clear and the controller must initiate a 5-second "direction switch" procedure. After 3 consecutive aircraft using one direction, the runway must switch directions to accommodate aircraft waiting for the other direction (if any are waiting). Aircraft waiting for the current runway direction should be prioritized over aircraft waiting for the opposite direction, unless a direction switch is required.

**Requirement 8:** Each aircraft has a fuel reserve randomly assigned between 20-60 seconds (at creation time). If an aircraft waits in the holding pattern longer than its fuel reserve, it must declare a fuel emergency. Fuel emergency aircraft must be granted immediate runway access, overriding normal rotation rules and even emergency aircraft priority. However, fuel emergency aircraft still respect the runway capacity limit (2 aircraft maximum) and cannot interrupt the controller's break. The controller must track aircraft fuel levels and prioritize those approaching fuel exhaustion to prevent crashes.

**Requirement 9:** Your code shall not deadlock.

## Non-Functional Requirements

**Requirement 10:** Your source file shall be named `runway.c`. The source file must be ASCII text files. No binary submissions will be accepted.

**Requirement 11:** Tabs or spaces shall be used to indent the code. Your code must use one or the other. All indentation must be consistent.

**Requirement 12:** No line of code shall exceed 100 characters.

**Requirement 13:** Each source code file shall have your name and student ID at the top. Do not remove the GPL copyright notice.

**Requirement 14**: All code must be well commented. This means descriptive comments that tell the intent of the code, not just what the code is executing.

**Requirement 15:** Keep your curly brace placement consistent. If you place curly braces on a new line, always place curly braces on a new line. Don't mix end line brace placement with new line brace placement.

**Requirement 16**: Each function should have a header that describes its name, any parameters expected, any return values, as well as a description of what the function does. For example:

```
/*
 * Function: commercial_enter
 * Parameters: aircraft_info - pointer to aircraft information structure
 * Returns: void
 * Description: Handles the synchronization for a commercial aircraft
 *              requesting runway access. Blocks until the aircraft
 *              can safely use the runway according to all constraints.
 */
```

**Requirement 17**: Remove all extraneous debug output before submission. The only output shall be the output of the simulation or any required logging messages.

**Requirement 18**: Your code shall compile cleanly on the GitHub codespace with no warnings using:

```
make
```

**Requirement 19**:  You shall not modify the compiler flags in the Makefile

**Requirement 20:** Your code shall not leak memory.  Use valgrind to verify your code is leak free.

```
valgrind --leak-check=full ./runway [test case]
```

**Requirement 21:** Removing or modifying a pre-existing `assert()` will result in a 0 for the assignment.

# Administrative

You have been provided a framework for the simulation, which creates a thread for the air traffic controller and a thread for each aircraft that wants to use the runway. You will need to add synchronization to ensure the above restrictions.

The source code for this assignment can be found on the course website at:

https://github.com/CSE3320-Fall-2025/Runway-Assignment

The aircraft threads are implemented in the functions `commercial_aircraft()`, `cargo_aircraft()`, and `emergency_aircraft()`, which simulate commercial, cargo, and emergency flights, respectively. After being created, a commercial aircraft executes three functions: it requests runway access (`commercial_enter()`), it uses the runway (`use_runway()`), and then clears the runway (`commercial_leave()`). Cargo and emergency aircraft call the corresponding functions `cargo_enter()`, `use_runway()`, `cargo_leave()` and `emergency_enter()`, `use_runway()`, `emergency_leave()`. All synchronization between threads should be added in the `..._enter()` and `..._leave()` functions of the aircraft and the function `controller_thread()`, which implements the air traffic controller.

The simulation framework is implemented in the file `runway.c`. The program expects as an argument the name of an input file which controls the simulation. The input file specifies the

arrival of aircraft and the amount of time aircraft spend using the runway. More precisely, the file has one line for each aircraft containing three numbers. The first number is the time (in seconds) between the arrival of this aircraft and the previous aircraft. The second number is the number of seconds the aircraft needs to spend using the runway. The third number is the aircraft type (0 = commercial, 1 = cargo, 2 = emergency). Each aircraft will also be assigned a random fuel reserve between 20-60 seconds at creation time.

The provided code implements all the functionality for the aircraft and the controller, but does not implement any synchronization. To help you in developing your code, a number of assert statements that help you check for correctness have been added. **DO NOT delete those assert statements**. Also, do not make any changes to the functions `use_runway`, `take_break`, and `switch_direction`. You might want to add additional assert statements, for example for ensuring that the number of aircraft since the last break is less than the limit, ensuring that there are not commercial and cargo aircraft using the runway at the same time, verifying runway direction consistency, or checking that fuel emergency aircraft are properly prioritized.

Before you start your work, compile `runway.c` and try to run it on this sample input file `test-cases/test01_simple.txt. This will simulate different aircraft using the runway with various types, durations, and arrival times.

```
./runway test-cases/test01_simple.txt
```

**This assignment must be coded in C. Any other language will result in 0 points.** Your programs will be compiled and graded on omega.uta.edu. Please make sure they compile and run on the GitHub Codespace before submitting them. Code that does not compile with:

```
make
```

will result in a 0.

# Design Considerations

This assignment includes several features that significantly increase the complexity and potential for deadlock compared to simpler synchronization problems:

1. **Multiple Priority Levels:** You must handle regular aircraft, emergency aircraft, and fuel emergency aircraft with different priority levels. Ensure that high-priority aircraft don't starve low-priority ones indefinitely.

2. **Dynamic Priorities**: Aircraft fuel decreases over time, potentially elevating their priority to fuel emergency status. You'll need to track waiting times and fuel levels.

3. **Runway Direction State:** The runway direction is a shared state that requires coordination. Direction switches require the runway to be completely empty, which can create convoy effects.

4. **Time Constraints:** The 30-second emergency response requirement and fuel reserve depletion create real-time constraints that your synchronization must respect.

5. **Break Scheduling:** The controller's break schedule must be respected even during emergencies, requiring careful state management.

**Hints for avoiding deadlock:**
- Consider using condition variables with timeouts for fuel monitoring
- Implement a priority queue or similar structure for waiting aircraft
- Be careful about lock ordering when checking multiple conditions
- The direction switch procedure is particularly deadlock-prone - ensure all aircraft release locks during the switch
- Test with input files that stress-test edge cases (many emergencies, rapid fuel depletion, frequent direction changes)

Your program, `runway.c`, is to be turned in via using to the main branch of your GitHub repo. Submission time is determined by the GitHub time. You may submit your programs as often as you wish. Only your last submission will be graded.

****You may use no other outside code.****

# Academic Integrity

This assignment must be 100% your own work. No code may be copied from friends, previous students, books, web pages, etc. All code submitted is automatically checked against a database of previous semester's graded assignments, current student's code and common web sources.  You may use any code I provide to you including code in the Code-Samples repository.

No LLM or AI assistants or agents may be used.

By submitting your code on GitHub you are attesting that you have neither given nor received unauthorized assistance on this work. **Code that is copied from an external source or used as inspiration, excluding the course GitHub or Canvas, will result in a 0 for the assignment and referral to the Office of Student Conduct.**